МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**РУТ (МИИТ)**

**Кафедра «Управление и защита информации»**

КУРСОВОЙ ПРОЕКТ

по дисциплине

«Системы искусственого интеллекта»

на тему:

*«Решение задачи регрессии: предсказание медицинских расходов»*

Выполнил: студент гр. УВП-412 Рогов К.Д.

Принял: доцент Зольникова Н. Н.

Москва – 2025

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc191295926)

[1. Предварительная обработка данных и постановка задачи 4](#_Toc191295927)

[1.1 Описание набора данных 4](#_Toc191295928)

[1.2 Подготовка данных 4](#_Toc191295929)

[2. Реализация регрессии двумя способами 7](#_Toc191295930)

[2.1 Модель без использования библиотек 7](#_Toc191295931)

[2.2 Модель с использованием библиотек 12](#_Toc191295932)

[Сохранение параметров модели 14](#_Toc191295933)

[3. Разработка программы для работы с обученной моделью 14](#_Toc191295934)

[ЗАКЛЮЧЕНИЕ 16](#_Toc191295935)

# **ВВЕДЕНИЕ**

В данной работе реализована задача предсказания страховых расходов на основе набора данных «insurance». Решение задачи выполнено двумя способами:

* **Модель без использования библиотек** – реализована с нуля с помощью метода градиентного спуска.
* **Модель с использованием библиотек** – использует готовые инструменты, в частности класс LinearRegression из scikit-learn.

Кроме того, разработана программа с графическим интерфейсом (GUI), которая позволяет пользователю ввести исходные данные и получить предсказание расходов. Отчет включает анализ предварительной обработки данных, описание моделей, визуализацию результатов и разъяснение нюансов реализации.

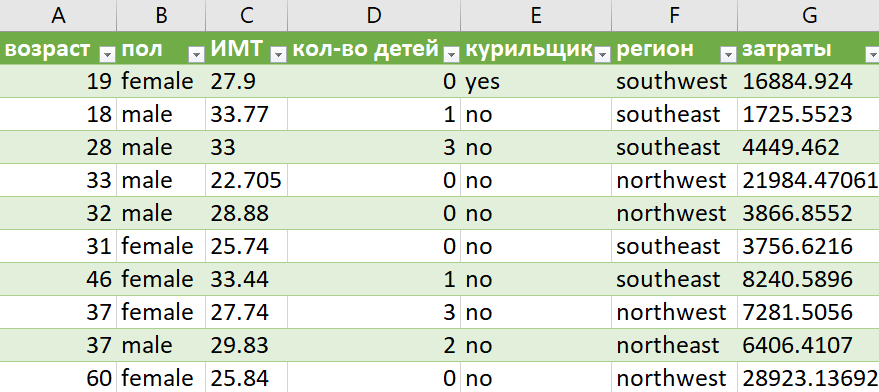
# **1. Предварительная обработка данных и постановка задачи**

## 1.1 Описание набора данных

Набор данных «insurance.csv» содержит информацию о клиентах и их страховых расходах. Основные признаки включают:

* **age** – возраст клиента,
* **sex** – пол (значения «male» и «female»),
* **bmi** – индекс массы тела,
* **children** – количество детей,
* **smoker** – наличие привычки курения («yes» или «no»),
* **region** – регион проживания.

Целевая переменная – **charges** – представляет собой затраты на страхование. В целях стабилизации разброса значений и улучшения качества модели, значения колонки преобразуются с помощью логарифма.



## 1.2 Подготовка данных

Для корректной работы моделей машинного обучения необходимо правильно подготовить данные. Рассмотрим основные этапы предобработки.

#### 1. Кодирование категориальных признаков

Категориальные переменные, такие как sex, smoker и region, должны быть преобразованы в числовые значения, чтобы их можно было использовать в моделях машинного обучения:

# Кодирование бинарных категориальных признаков

df['sex'] = df['sex'].map({'male': 0, 'female': 1})

df['smoker'] = df['smoker'].map({'no': 0, 'yes': 1})

# Кодирование региона через числовые индексы

unique\_regions = df['region'].unique()

region\_mapping = {region: i for i, region in enumerate(unique\_regions)}

df['region'] = df['region'].map(region\_mapping)

Преобразование категориальных признаков делает их интерпретируемыми для моделей и улучшает качество предсказаний.

#### Нормализация числовых признаков

Нормализация (стандартизация) помогает избежать влияния масштабов разных переменных и ускоряет обучение модели. Используем стандартизацию:

for col in ['age', 'bmi', 'children']:

mean\_val = df[col].mean()

std\_val = df[col].std()

df[col] = (df[col] - mean\_val) / std\_val

* Позволяет модели быстрее обучаться, так как значения находятся в одном масштабе.
* Улучшает интерпретируемость коэффициентов в линейных моделях.
* Уменьшает влияние признаков с большим разбросом.

#### Логарифмическое преобразование целевой переменной

Так как распределение charges может быть сильно скошенным, применяем логарифмирование, чтобы сделать его более нормальным:

df['charges'] = np.log(df['charges'])

* Уменьшает влияние выбросов.
* Улучшает распределение данных, что повышает качество предсказаний.

#### Перемешивание и разбиение данных

Для равномерного распределения данных между обучающей и тестовой выборкой перемешаем данные перед разделением:

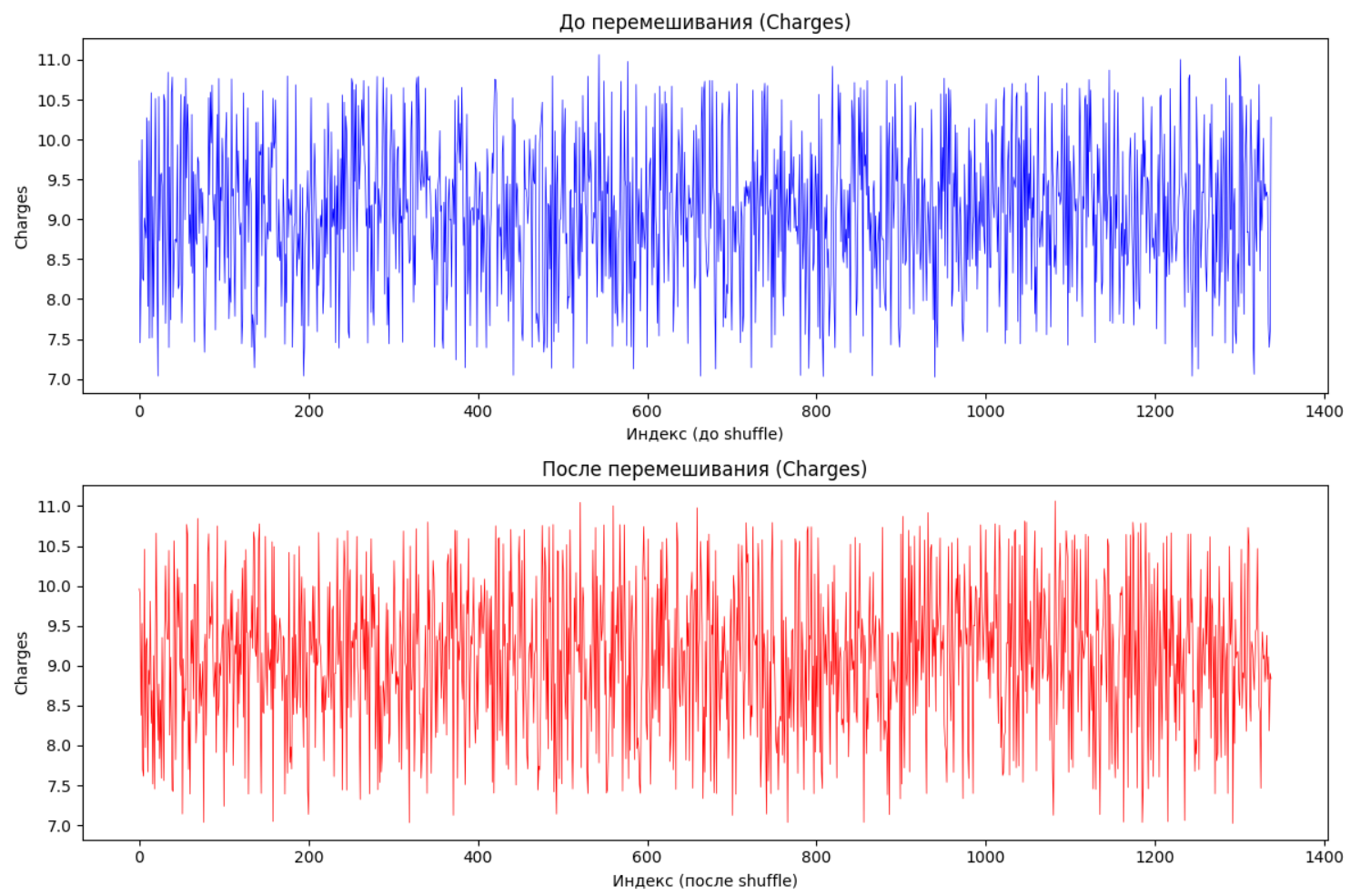
for i in range(len(df\_list) - 1, 0, -1):  
 j = random.randint(0, i)  
 df\_list[i], df\_list[j] = df\_list[j], df\_list[i]  
  
df = pd.DataFrame(df\_list, columns=df.columns)  
  
df["shuffled\_index"] = list(range(len(df)))  
  
# Split manually into train and test sets (70% train, 30% test)  
split\_point = int(0.7 \* len(df))

#### Визуализация этапа предобработки

Для контроля корректности перемешивания и нормализации строим графики.

##### Эффект перемешивания

Показывает, как изменился порядок данных после shuffle:



##### Распределение признаков до и после нормализации

Сравниваем распределение переменных age и bmi до и после нормализации:

fig, axes = plt.subplots(1, 2, figsize=(12, 10))

columns\_to\_plot = ['age', 'bmi']

for i, col in enumerate(columns\_to\_plot):

ax = axes[i]

ax.hist(df\_before\_shuffle[col], bins=20, alpha=0.5, label="До нормализации")

ax.hist(df[col], bins=20, alpha=0.5, label="После нормализации", color='red')

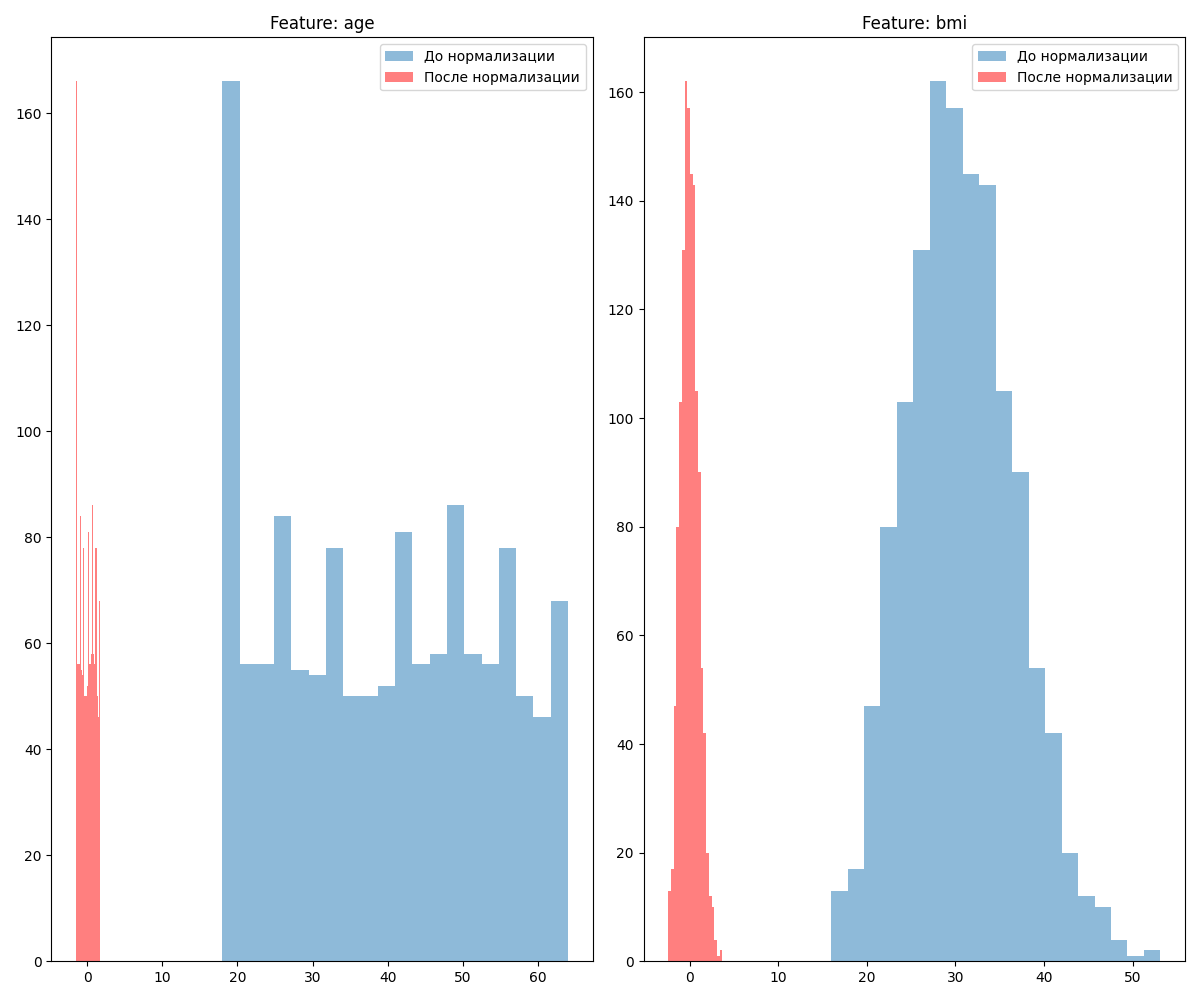
ax.set\_title(f"Feature: {col}")

ax.legend()

plt.tight\_layout()

plt.savefig("preprocessing\_effect.png")

plt.close()



Файлы с подготовленными данными сохранены:

* train\_numeric.csv (обучающая выборка)
* test\_numeric.csv (тестовая выборка)

# **2. Реализация регрессии двумя способами**

## 2.1 Модель без использования библиотек

В файле model\_no\_libs.py реализована линейная регрессия «с нуля» с использованием метода градиентного спуска. Мы выполняем все вычисления вручную, используя стандартные структуры Python (списки). Такой подход полезен для глубокого понимания основ машинного обучения и работы алгоритма градиентного спуска.

Основные этапы:

* **Добавление столбца смещения (bias).**

Для учета свободного члена к матрице признаков добавляется столбец единиц. Чтобы учесть его в вычислениях, добавляем в матрицу признаков столбец единиц.

def add\_bias(X):

return [[1] + row for row in X]

X\_train\_bias = add\_bias(X\_train)

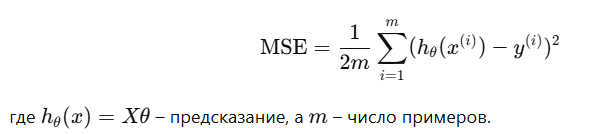
X\_test\_bias = add\_bias(X\_test)

* **Инициализация параметров.**  
  На старте мы инициализируем вектор коэффициентов модели theta. Это значения, которые будут обновляться в процессе обучения. Модель линейной регрессии определяет веса (коэффициенты) θ\thetaθ, которые обучаются с помощью градиентного спуска. Эти коэффициенты инициализируются случайными значениями, обычно небольшими, чтобы избежать сильных колебаний в начале обучения:

num\_features = len(X\_train\_bias[0]) # Количество признаков (включая bias)

theta = [[random.uniform(-0.1, 0.1)] for \_ in range(num\_features)] # Инициализация вектора параметров случайными значениями от -0.1 до 0.1

* **Определение функции ошибки (MSE) и вычисление градиента.**  
  Функция ошибки Mean Squared Error (MSE) показывает, насколько предсказания модели отклоняются от реальных значений.Функция ошибки определяется как:



Код функции:

’’’Вычисляет среднеквадратичную ошибку (MSE) для заданных X, y и параметров theta. Args: X (list of list): Матрица признаков y (list of list): Вектор истинных значений theta (list of list): Вектор параметров модели’’’

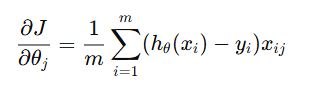
def mse(X, y, theta):

predictions = mat\_mult(X, theta)

error = matrix\_subtract(predictions, y)

return sum(row[0] \*\* 2 for row in error) / (2 \* len(X))

Аналогично определяется функция для вычисления градиента. Градиент MSE позволяет вычислять направление, в котором следует изменять параметры модели:



def gradient(X, y, theta, lambda\_reg=0.01):

predictions = mat\_mult(X, theta)

error = matrix\_subtract(predictions, y)

X\_T = transpose(X)

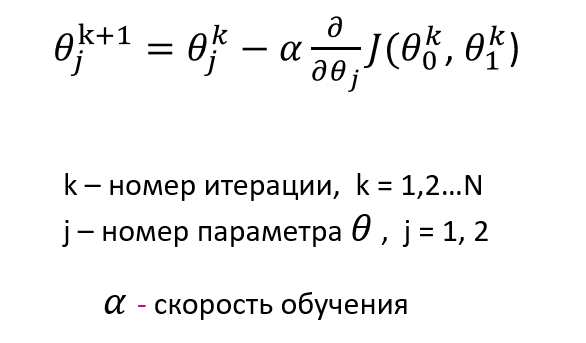
grad = mat\_mult(X\_T, error)

for j in range(len(theta)):

grad[j][0] += lambda\_reg \* theta[j][0]

return scalar\_mult\_matrix(1 / len(X), grad)

* **Градиентный спуск.**  
  В цикле происходит обновление параметров. Градиентный спуск итеративно обновляет параметры модели, следуя в сторону уменьшения MSE:



for i in range(num\_iterations):

grad = gradient(X\_train\_bias, y\_train, theta) # Вычисляем градиент

for j in range(len(theta)):

theta[j][0] -= learning\_rate \* grad[j][0] # Обновляем параметры

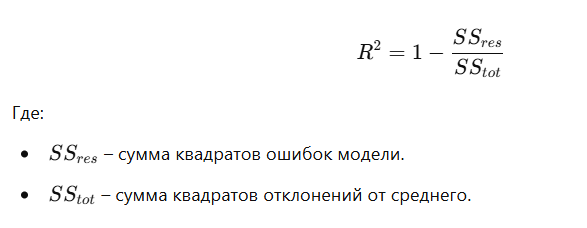
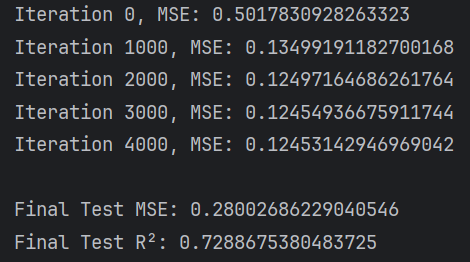
cost = mse(X\_train\_bias, y\_train, theta)

mse\_history.append(cost)

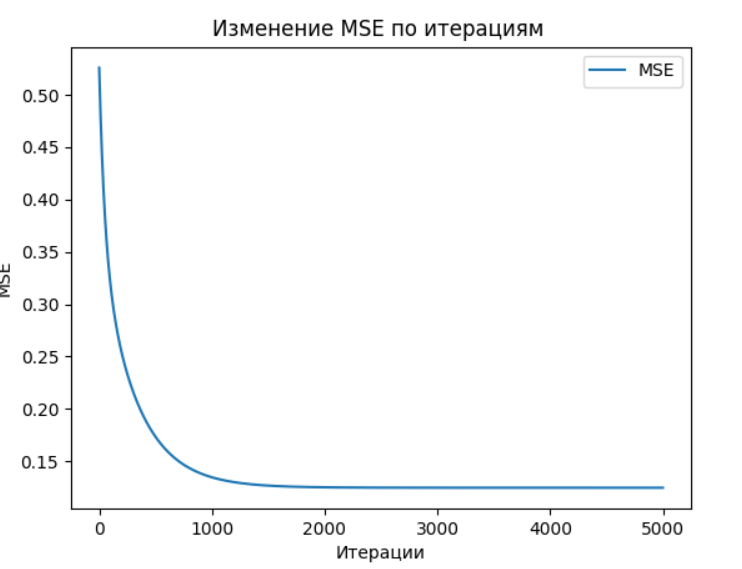
if i % 1000 == 0:

print(f"Iteration {i}, MSE: {cost}")

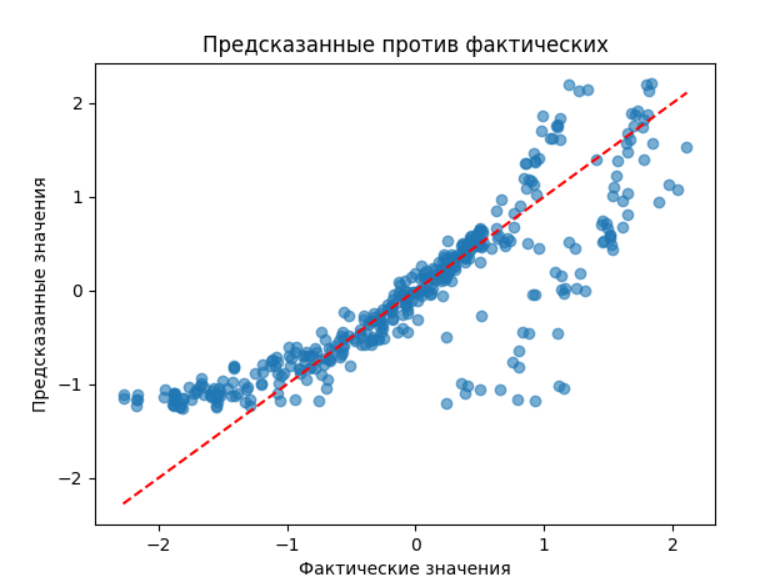
Каждые 1000 итераций выводится значение MSE для контроля процесса обучения.



* **Оценка качества модели.**  
  После обучения вычисляются показатели MSE и коэффициент детерминации R^2. Для визуализации результатов генерируются:
  + График изменения MSE по итерациям (no\_libs\_mse\_vs\_iteration.png).



* + График «Предсказанные против фактических» значений (predicted\_vs\_actual\_no\_libs.png).



## 2.2 Модель с использованием библиотек

Файл model\_libs.py демонстрирует применение библиотеки scikit-learn для построения модели линейной регрессии:

* **Загрузка данных.**  
  Данные считываются из файлов train\_numeric.csv и test\_numeric.csv, где уже произведена предварительная обработка.
* **Обучение модели с помощью LinearRegression.**

model = LinearRegression()

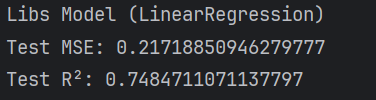
model.fit(X\_train, y\_train)

Этот готовый инструмент автоматически учитывает наличие смещения.

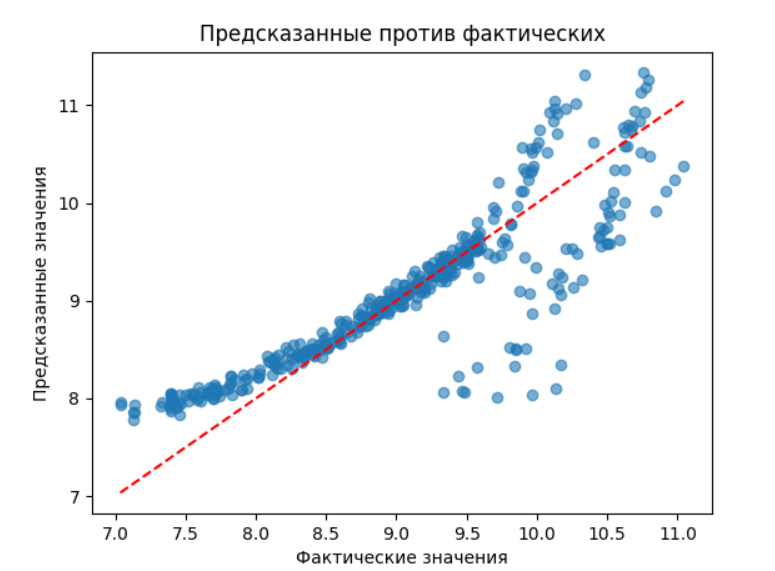
* **Оценка качества модели.**  
  Предсказания на тестовой выборке сравниваются с истинными значениями с помощью метрик MSE и R^2:

mse\_test = mean\_squared\_error(y\_test, y\_pred\_test)

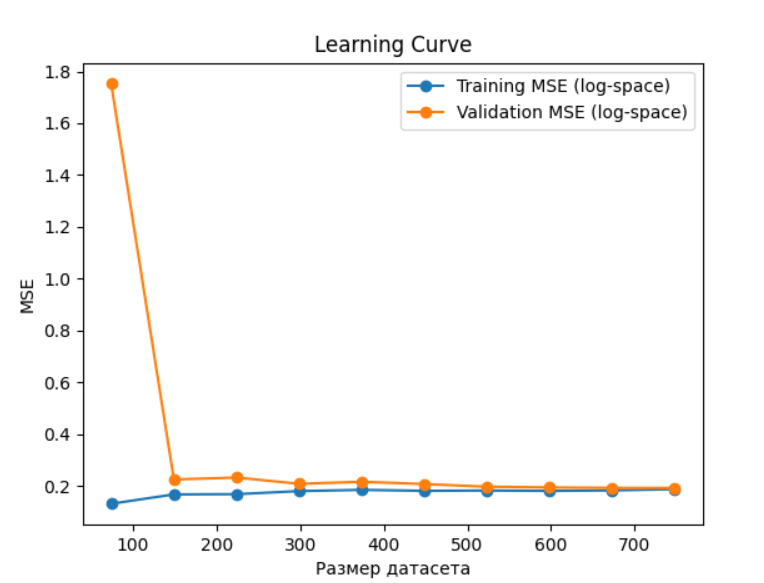
r2\_test = r2\_score(y\_test, y\_pred\_test)



* **Визуализация результатов.**  
  Для данной модели генерируются следующие графики:
  + **Predicted vs. Actual:**  
    Точечный график, показывающий соответствие предсказанных и фактических значений (файл predicted\_vs\_actual\_libs.png).



* **Learning Curve:**  
  Кривая обучения показывает, как ошибка (MSE) изменяется при увеличении размера обучающей выборки. Это помогает определить, достаточно ли данных для обучения модели.



## Сохранение параметров модели

После обучения модели её параметры (веса) сохраняются для дальнейшего использования.

# Сохранение коэффициентов модели

theta\_libs = np.hstack([model.intercept\_, model.coef\_])

np.savetxt("theta\_libs.txt", theta\_libs)

print("Saved model parameters to theta\_libs.txt.")

* model.intercept\_ – свободный член (смещение).
* model.coef\_ – коэффициенты линейной регрессии.

# **3. Разработка программы для работы с обученной моделью**

Файл predict\_ui.py реализует графический интерфейс для взаимодействия с пользователем. Основные моменты реализации:

* **Предварительные расчёты на базе исходных данных.**  
  Считываются исходные данные из файла insurance.csv для вычисления статистических параметров (медианы, средних значений и стандартных отклонений), которые используются для нормализации входных данных.
* **Кодирование категориальных признаков.**  
  Значения для признаков sex, smoker и region преобразуются в числовые с использованием заранее определённых отображений.
* **Загрузка параметров обученной модели.**  
  Из файла theta\_libs.txt загружаются коэффициенты, полученные в модели с использованием библиотек.

theta\_libs = np.loadtxt("theta\_libs.txt")

* **Функция предсказания.**  
  Функция predict\_charge принимает сырые входные данные, проводит их нормализацию и кодирование, формирует вектор признаков (с учетом смещения) и вычисляет логарифмическое предсказание, которое затем преобразуется в значение в долларах:

def predict\_charge(raw\_age, raw\_sex, raw\_bmi, raw\_children, raw\_smoker, raw\_region):

norm\_age = (raw\_age - mean\_age) / std\_age

norm\_bmi = (raw\_bmi - mean\_bmi) / std\_bmi

norm\_children = (raw\_children - mean\_children) / std\_children

mapped\_sex = sex\_map[raw\_sex.lower()]

mapped\_smoker = smoker\_map[raw\_smoker.lower()]

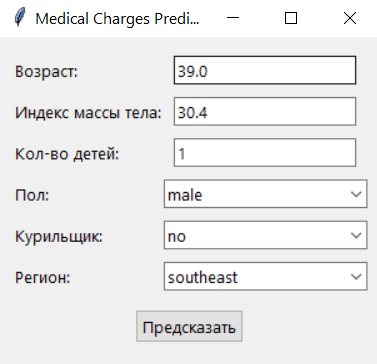
mapped\_region = region\_mapping[raw\_region]

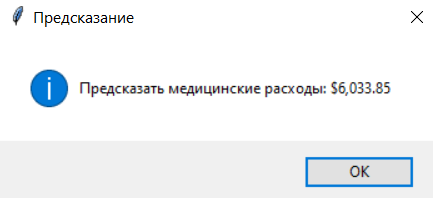
x = np.array([1.0, norm\_age, mapped\_sex, norm\_bmi, norm\_children, mapped\_smoker, mapped\_region], dtype=float)

log\_pred = np.dot(theta\_libs, x)

return np.exp(log\_pred)

* **Интерфейс пользователя.**  
  С использованием библиотеки tkinter создается окно с полями для ввода значений (возраст, BMI, количество детей, выбор пола, курильщика и региона). При нажатии кнопки «Предсказать» данные обрабатываются, и пользователю выводится предсказанное значение расходов.





# **ЗАКЛЮЧЕНИЕ**

В отчете описан комплексный подход к задаче предсказания страховых расходов. Были реализованы две модели:

* **Без использования библиотек:** модель, реализованная вручную, демонстрирует принцип работы градиентного спуска и позволяет глубже понять внутренние механизмы линейной регрессии.
* **С использованием библиотек:** применение scikit-learn позволяет быстро построить модель и получить готовые решения для оценки качества, а также визуализировать результаты.

Кроме того, разработанный графический интерфейс делает систему доступной для конечного пользователя, позволяя вводить данные и получать предсказания в режиме реального времени.

Предоставленные визуализации (например, графики предсказанных против фактических значений, learning curve и эффекты предварительной обработки) помогают оценить качество моделей и увидеть влияние этапов обработки данных на итоговый результат. Такой комплексный подход даёт возможность не только построить модель, но и понять все нюансы её работы, а также выявить возможные направления для дальнейшего улучшения.

**ПРИЛОЖЕНИЕ**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import random  
  
# Load the raw insurance dataset  
df = pd.read\_csv("insurance.csv")  
  
# Store a copy before shuffling for visualization  
df\_before\_shuffle = df.copy()  
  
# Assign an index column before shuffling to track movement  
df\_before\_shuffle["original\_index"] = list(range(len(df)))  
  
# -------------------------------  
# 1. Label-encode categorical columns  
# -------------------------------  
df['sex'] = df['sex'].map({'male': 0, 'female': 1})  
df['smoker'] = df['smoker'].map({'no': 0, 'yes': 1})  
unique\_regions = df['region'].unique()  
region\_mapping = {region: i for i, region in enumerate(unique\_regions)}  
df['region'] = df['region'].map(region\_mapping)  
  
# -------------------------------  
# 2. Normalize numeric features  
# -------------------------------  
for col in ['age', 'bmi', 'children']:  
 mean\_val = df[col].mean()  
 std\_val = df[col].std()  
 df[col] = (df[col] - mean\_val) / std\_val  
  
# -------------------------------  
# 3. Log-transform the target  
# -------------------------------  
df['charges'] = df['charges'].apply(lambda x: x if x <= 0 else np.log(x))  
df\_before\_shuffle['charges'] = df\_before\_shuffle['charges'].apply(lambda x: x if x <= 0 else np.log(x))  
  
# -------------------------------  
# 4. Manually Shuffle the Data (Fisher-Yates Algorithm)  
# -------------------------------  
random.seed(42) # Ensure reproducibility  
df\_list = df.values.tolist()  
  
for i in range(len(df\_list) - 1, 0, -1):  
 j = random.randint(0, i)  
 df\_list[i], df\_list[j] = df\_list[j], df\_list[i]  
  
df = pd.DataFrame(df\_list, columns=df.columns)  
  
df["shuffled\_index"] = list(range(len(df)))  
  
# Split manually into train and test sets (70% train, 30% test)  
split\_point = int(0.7 \* len(df))  
train\_df = df.iloc[:split\_point].copy()  
test\_df = df.iloc[split\_point:].copy()  
  
# Save as CSV  
train\_df.to\_csv("train\_numeric.csv", index=False)  
test\_df.to\_csv("test\_numeric.csv", index=False)  
  
# -------------------------------  
# Improved Visualization: Shuffle Effect with Line Plots  
# -------------------------------  
plt.figure(figsize=(12, 8))  
  
# Plot before shuffle as a line plot  
plt.subplot(2, 1, 1)  
plt.plot(df\_before\_shuffle["original\_index"], df\_before\_shuffle["charges"], color="blue", linestyle="-", linewidth=0.5)  
plt.title("До перемешивания (Charges)")  
plt.xlabel("Индекс (до shuffle)")  
plt.ylabel("Charges")  
  
# Plot after shuffle as a line plot  
plt.subplot(2, 1, 2)  
plt.plot(df["shuffled\_index"], df["charges"], color="red", linestyle="-", linewidth=0.5)  
plt.title("После перемешивания (Charges)")  
plt.xlabel("Индекс (после shuffle)")  
plt.ylabel("Charges")  
  
plt.tight\_layout()  
plt.savefig("shuffle\_effect\_lineplot.png")  
plt.close()  
  
# -------------------------------  
# Visualization: Feature Distribution Before and After Preprocessing  
# -------------------------------  
fig, axes = plt.subplots(1, 2, figsize=(12, 10))  
  
columns\_to\_plot = ['age', 'bmi']  
for i, col in enumerate(columns\_to\_plot):  
 ax = axes[i]  
 ax.hist(df\_before\_shuffle[col], bins=20, alpha=0.5, label="До нормализации")  
 ax.hist(df[col], bins=20, alpha=0.5, label="После нормализации", color='red')  
 ax.set\_title(f"Feature: {col}")  
 ax.legend()  
  
plt.tight\_layout()  
plt.savefig("preprocessing\_effect.png")  
plt.close()  
  
print("Preprocessing complete. Numeric files saved:")  
print(" - train\_numeric.csv")  
print(" - test\_numeric.csv")  
print(" - shuffle\_effect\_lineplot.png (clearer shuffle visualization with line plots)")  
print(" - preprocessing\_effect.png (feature preprocessing visualization)")

# File: model\_libs.py  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error, r2\_score  
from sklearn.model\_selection import learning\_curve  
  
# -----------------------------  
# 1. Load numeric train/test data (all numeric, log-target)  
# -----------------------------  
train\_df = pd.read\_csv("train\_numeric.csv")  
test\_df = pd.read\_csv("test\_numeric.csv")  
  
X\_train = train\_df.drop("charges", axis=1)  
y\_train = train\_df["charges"] # log-transformed  
X\_test = test\_df.drop("charges", axis=1)  
y\_test = test\_df["charges"]  
  
# -----------------------------  
# 2. Fit scikit-learn LinearRegression  
# -----------------------------  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
y\_pred\_test = model.predict(X\_test)  
  
# -----------------------------  
# 3. Compute MSE & R² in log-space  
# -----------------------------  
mse\_test = mean\_squared\_error(y\_test, y\_pred\_test)  
r2\_test = r2\_score(y\_test, y\_pred\_test)  
  
print("Libs Model (LinearRegression)")  
print("Test MSE:", mse\_test)  
print("Test R²:", r2\_test)  
  
# -----------------------------  
# 4. Plot Predicted vs. Actual (log-space)  
# -----------------------------  
plt.figure()  
plt.scatter(y\_test, y\_pred\_test, alpha=0.6)  
plt.plot([y\_test.min(), y\_test.max()],  
 [y\_test.min(), y\_test.max()],  
 'r--')  
plt.xlabel("Фактические значения")  
plt.ylabel("Предсказанные значения")  
plt.title("Предсказанные против фактических")  
plt.savefig("predicted\_vs\_actual\_libs.png")  
plt.close()  
  
# -----------------------------  
# 5. Generate a Learning Curve  
# -----------------------------  
from sklearn.model\_selection import learning\_curve  
  
train\_sizes, train\_scores, val\_scores = learning\_curve(  
 model, X\_train, y\_train,  
 scoring="neg\_mean\_squared\_error",  
 cv=5,  
 train\_sizes=np.linspace(0.1, 1.0, 10),  
 random\_state=42  
)  
  
# Convert negative MSE to positive MSE  
train\_mse = -np.mean(train\_scores, axis=1)  
val\_mse = -np.mean(val\_scores, axis=1)  
  
plt.figure()  
plt.plot(train\_sizes, train\_mse, 'o-', label="Training MSE (log-space)")  
plt.plot(train\_sizes, val\_mse, 'o-', label="Validation MSE (log-space)")  
plt.xlabel("Размер датасета")  
plt.ylabel("MSE")  
plt.title("Learning Curve")  
plt.legend()  
plt.savefig("libs\_learning\_curve.png")  
plt.close()  
  
# -----------------------------  
# 6. Save learned parameters  
# -----------------------------  
theta\_libs = np.hstack([model.intercept\_, model.coef\_])  
np.savetxt("theta\_libs.txt", theta\_libs)  
print("Saved libs model parameters to theta\_libs.txt.")  
print("Plots saved: predicted\_vs\_actual\_libs.png, libs\_learning\_curve.png")

# File: model\_no\_numpy.py  
import pandas as pd  
import matplotlib.pyplot as plt  
import random  
  
# -----------------------------  
# 1. Load numeric train/test data  
# -----------------------------  
train\_df = pd.read\_csv("train\_numeric.csv")  
test\_df = pd.read\_csv("test\_numeric.csv")  
  
# -----------------------------  
# 2. Compute Mean & Std for Normalization  
# -----------------------------  
def compute\_mean\_std(column):  
 *"""Compute mean and standard deviation for normalization."""* mean = sum(column) / len(column)  
 variance = sum((x - mean) \*\* 2 for x in column) / len(column)  
 std\_dev = variance \*\* 0.5  
 return mean, std\_dev if std\_dev != 0 else 1 # Prevent division by zero  
  
# Normalize numeric features  
for col in ['age', 'bmi', 'children']:  
 mean\_val, std\_val = compute\_mean\_std(train\_df[col].tolist())  
 train\_df[col] = [(x - mean\_val) / std\_val for x in train\_df[col]]  
 test\_df[col] = [(x - mean\_val) / std\_val for x in test\_df[col]]  
  
# Normalize target variable (log-space)  
y\_mean, y\_std = compute\_mean\_std(train\_df["charges"].tolist())  
train\_df["charges"] = [(x - y\_mean) / y\_std for x in train\_df["charges"]]  
test\_df["charges"] = [(x - y\_mean) / y\_std for x in test\_df["charges"]]  
  
# -----------------------------  
# 3. Select Only Relevant Features  
# -----------------------------  
selected\_features = ["age", "bmi", "smoker"]  
X\_train = train\_df[selected\_features].values.tolist()  
y\_train = [[val] for val in train\_df["charges"].tolist()]  
X\_test = test\_df[selected\_features].values.tolist()  
y\_test = [[val] for val in test\_df["charges"].tolist()]  
  
# -----------------------------  
# 4. Add Bias Column  
# -----------------------------  
def add\_bias(X):  
 return [[1] + row for row in X]  
  
X\_train\_bias = add\_bias(X\_train)  
X\_test\_bias = add\_bias(X\_test)  
  
# -----------------------------  
# 5. Initialize Parameters  
# -----------------------------  
num\_features = len(X\_train\_bias[0])  
theta = [[random.uniform(-0.1, 0.1)] for \_ in range(num\_features)]  
  
# -----------------------------  
# 6. Define Matrix Operations  
# -----------------------------  
def mat\_mult(A, B):  
 *"""Multiply matrix A (m x n) with matrix B (n x p)."""* m, n = len(A), len(A[0])  
 p = len(B[0])  
 result = [[sum(A[i][k] \* B[k][j] for k in range(n)) for j in range(p)] for i in range(m)]  
 return result  
  
def transpose(A):  
 *"""Transpose a matrix."""* return [[A[i][j] for i in range(len(A))] for j in range(len(A[0]))]  
  
def scalar\_mult\_matrix(s, A):  
 *"""Multiply each element in matrix A by scalar s."""* return [[s \* val for val in row] for row in A]  
  
def matrix\_subtract(A, B):  
 *"""Subtract matrix B from A element-wise."""* return [[A[i][j] - B[i][j] for j in range(len(A[0]))] for i in range(len(A))]  
  
# -----------------------------  
# 7. Define MSE & Gradient  
# -----------------------------  
def mse(X, y, theta):  
 *"""Compute Mean Squared Error (MSE)."""* m = len(X)  
 predictions = mat\_mult(X, theta)  
 error = matrix\_subtract(predictions, y)  
 squared\_error = [[val[0] \*\* 2] for val in error]  
 return sum(row[0] for row in squared\_error) / (2 \* m)  
  
def gradient(X, y, theta, lambda\_reg=0.01):  
 *"""Compute the gradient of MSE with L2 regularization."""* m = len(X)  
 predictions = mat\_mult(X, theta)  
 error = matrix\_subtract(predictions, y)  
 X\_T = transpose(X)  
 grad = mat\_mult(X\_T, error)  
  
 # Add L2 regularization  
 for j in range(len(theta)):  
 grad[j][0] += lambda\_reg \* theta[j][0]  
  
 return scalar\_mult\_matrix(1 / m, grad)  
  
# -----------------------------  
# 8. Gradient Descent  
# -----------------------------  
learning\_rate = 1e-2  
num\_iterations = 5000  
mse\_history = []  
  
for i in range(num\_iterations):  
 grad = gradient(X\_train\_bias, y\_train, theta)  
  
 for j in range(len(theta)):  
 theta[j][0] -= learning\_rate \* grad[j][0]  
  
 cost = mse(X\_train\_bias, y\_train, theta)  
 mse\_history.append(cost)  
  
 if i % 1000 == 0:  
 print(f"Iteration {i}, MSE: {cost}")  
  
# -----------------------------  
# 9. Final MSE & R² on Test Set  
# -----------------------------  
y\_pred\_test = mat\_mult(X\_test\_bias, theta)  
test\_mse\_log = 2 \* mse(X\_test\_bias, y\_test, theta)  
  
# Compute R² manually  
def mean(values):  
 *"""Compute mean of a list."""* return sum(values) / len(values)  
  
y\_test\_vals = [row[0] for row in y\_test]  
y\_pred\_vals = [row[0] for row in y\_pred\_test]  
mean\_y\_test = mean(y\_test\_vals)  
  
sst\_test = sum((val - mean\_y\_test) \*\* 2 for val in y\_test\_vals)  
ssr\_test = sum((y\_pred\_vals[i] - y\_test\_vals[i]) \*\* 2 for i in range(len(y\_test\_vals)))  
r2\_test = 1 - (ssr\_test / sst\_test)  
  
print("\nFinal Test MSE:", test\_mse\_log)  
print("Final Test R²:", r2\_test)  
  
# -----------------------------  
# 10. Plot MSE vs. Iterations  
# -----------------------------  
plt.figure()  
plt.plot(mse\_history, label="MSE")  
plt.xlabel("Итерации")  
plt.ylabel("MSE")  
plt.title("Изменение MSE по итерациям ")  
plt.legend()  
plt.savefig("no\_libs\_mse\_vs\_iteration.png")  
plt.close()  
  
# -----------------------------  
# 11. Predicted vs. Actual (log-space)  
# -----------------------------  
plt.figure()  
plt.scatter(y\_test\_vals, y\_pred\_vals, alpha=0.6)  
plt.plot([min(y\_test\_vals), max(y\_test\_vals)],  
 [min(y\_test\_vals), max(y\_test\_vals)],  
 'r--')  
plt.xlabel("Фактические значения")  
plt.ylabel("Предсказанные значения")  
plt.title("Предсказанные против фактических")  
plt.savefig("predicted\_vs\_actual\_no\_libs.png")  
plt.close()  
  
# -----------------------------  
# 12. Save learned theta (without NumPy)  
# -----------------------------  
with open("theta\_no\_numpy.txt", "w") as f:  
 for row in theta:  
 f.write(str(row[0]) + "\n")  
  
print("Saved no-libs model parameters to theta\_no\_numpy.txt.")  
print("Plots saved: no\_libs\_mse\_vs\_iteration.png, predicted\_vs\_actual\_no\_libs.png")

# File: predict\_ui.py  
import tkinter as tk  
from tkinter import ttk, messagebox  
import pandas as pd  
import numpy as np  
  
# ----------------------------------------------------------------------------  
# 1. Load the raw dataset to compute defaults and transformation parameters  
# ----------------------------------------------------------------------------  
df = pd.read\_csv("insurance.csv")  
  
# Compute default values from raw data  
default\_age = df['age'].median()  
default\_bmi = df['bmi'].median()  
default\_children = int(df['children'].median()) # you may choose mode if preferred  
default\_sex = df['sex'].mode()[0] # 'male' or 'female'  
default\_smoker = df['smoker'].mode()[0] # 'no' or 'yes'  
default\_region = df['region'].mode()[0]  
  
# Compute normalization parameters for numeric features (as used in preprocess\_split.py)  
mean\_age = df['age'].mean()  
std\_age = df['age'].std()  
  
mean\_bmi = df['bmi'].mean()  
std\_bmi = df['bmi'].std()  
  
mean\_children = df['children'].mean()  
std\_children = df['children'].std()  
  
# ----------------------------------------------------------------------------  
# 2. Create categorical mappings exactly as in preprocess\_split.py  
# ----------------------------------------------------------------------------  
# For sex: male=0, female=1  
sex\_map = {'male': 0, 'female': 1}  
  
# For smoker: no=0, yes=1  
smoker\_map = {'no': 0, 'yes': 1}  
  
# For region: use the order in which they first appear in the dataset  
region\_mapping = {region: i for i, region in enumerate(df['region'].unique())}  
# Prepare a list of region names (in the same order as used in training)  
region\_list = list(region\_mapping.keys())  
  
# ----------------------------------------------------------------------------  
# 3. Load the learned model parameters (theta\_libs.txt) from the libs model  
# ----------------------------------------------------------------------------  
theta\_libs = np.loadtxt("theta\_libs.txt")  
# NOTE: The model was trained on the numeric dataset where features were:  
# [intercept, normalized age, encoded sex, normalized bmi, normalized children, encoded smoker, encoded region]  
  
# ----------------------------------------------------------------------------  
# 4. Define a prediction function that transforms raw inputs  
# ----------------------------------------------------------------------------  
def predict\_charge(raw\_age, raw\_sex, raw\_bmi, raw\_children, raw\_smoker, raw\_region):  
 *"""  
 Given raw (real-world) inputs, this function:  
 - Normalizes the numeric features using the training set parameters.  
 - Label-encodes categorical inputs.  
 - Forms the feature vector (with intercept) and applies the linear model.  
 - Exponentiates the log-prediction to return a dollar value.  
 """* # Normalize numeric features  
 norm\_age = (raw\_age - mean\_age) / std\_age  
 norm\_bmi = (raw\_bmi - mean\_bmi) / std\_bmi  
 norm\_children = (raw\_children - mean\_children) / std\_children  
  
 # Map categorical features  
 mapped\_sex = sex\_map[raw\_sex.lower()]  
 mapped\_smoker = smoker\_map[raw\_smoker.lower()]  
 mapped\_region = region\_mapping[raw\_region]  
  
 # Form feature vector (including intercept)  
 # Feature order: [1, norm\_age, mapped\_sex, norm\_bmi, norm\_children, mapped\_smoker, mapped\_region]  
 x = np.array([1.0, norm\_age, mapped\_sex, norm\_bmi, norm\_children, mapped\_smoker, mapped\_region], dtype=float)  
  
 # Compute log-charge prediction then exponentiate to get charge in dollars  
 log\_pred = np.dot(theta\_libs, x)  
 return np.exp(log\_pred)  
  
# ----------------------------------------------------------------------------  
# 5. Build the UI  
# ----------------------------------------------------------------------------  
root = tk.Tk()  
root.title("Medical Charges Predictor")  
  
frame = ttk.Frame(root, padding="10")  
frame.grid(row=0, column=0, sticky=(tk.W, tk.E))  
  
def create\_labeled\_entry(label\_text, row, default\_val):  
 ttk.Label(frame, text=label\_text).grid(row=row, column=0, sticky=tk.W, pady=5)  
 entry = ttk.Entry(frame)  
 entry.insert(0, str(default\_val))  
 entry.grid(row=row, column=1, pady=5)  
 return entry  
  
# Create entry fields for raw numeric inputs  
age\_entry = create\_labeled\_entry("Возраст:", 0, default\_age)  
bmi\_entry = create\_labeled\_entry("Индекс массы тела:", 1, default\_bmi)  
children\_entry = create\_labeled\_entry("Кол-во детей:", 2, default\_children)  
  
# For categorical values, use Comboboxes populated with raw labels.  
ttk.Label(frame, text="Пол:").grid(row=3, column=0, sticky=tk.W, pady=5)  
sex\_combo = ttk.Combobox(frame, values=["male", "female"], state="readonly")  
sex\_combo.set(default\_sex.lower())  
sex\_combo.grid(row=3, column=1, pady=5)  
  
ttk.Label(frame, text="Курильщик:").grid(row=4, column=0, sticky=tk.W, pady=5)  
smoker\_combo = ttk.Combobox(frame, values=["Да", "Нет"], state="readonly")  
smoker\_combo.set(default\_smoker.lower())  
smoker\_combo.grid(row=4, column=1, pady=5)  
  
ttk.Label(frame, text="Регион:").grid(row=5, column=0, sticky=tk.W, pady=5)  
region\_combo = ttk.Combobox(frame, values=region\_list, state="readonly")  
region\_combo.set(default\_region)  
region\_combo.grid(row=5, column=1, pady=5)  
  
def on\_predict():  
 try:  
 # Read raw inputs from UI  
 raw\_age = float(age\_entry.get())  
 raw\_bmi = float(bmi\_entry.get())  
 raw\_children = float(children\_entry.get())  
 raw\_sex = sex\_combo.get()  
 raw\_smoker = smoker\_combo.get()  
 raw\_region = region\_combo.get()  
  
 # Get prediction (in dollars)  
 prediction = predict\_charge(raw\_age, raw\_sex, raw\_bmi, raw\_children, raw\_smoker, raw\_region)  
 messagebox.showinfo("Предсказание", f"Предсказать медицинские расходы: ${prediction:,.2f}")  
 except Exception as e:  
 messagebox.showerror("Error", f"An error occurred: {e}")  
  
predict\_btn = ttk.Button(frame, text="Предсказать", command=on\_predict)  
predict\_btn.grid(row=6, column=0, columnspan=2, pady=10)  
  
root.mainloop()